10.2.9 Rechnen mit XPath 2.0

Programmiersprache für Berechnungen auf Folgen

Datentypen von XPath

- Folgen, deren Elemente entweder Atome oder Baumknoten sind
- Atome
  - Number (integer, decimal, float, double)
  - Boolean (`true()`, `false()`)
  - String (Unicode-Zeichen eingeschlossen in ' oder ")
  - XML-Schema Datentypen (später)

Konventionen für Werte

- Jeder Wert ist eine Folge
- Atome werden automatisch zu ein-elementigen Folgen konvertiert
- Viele Typkonversionen zwischen Atomen möglich
- Ein Knoten besitzt eine Identität (Referenz) und bezeichnet eindeutig einen Teilbaum des Dokuments
- Ein Folge wird *atomisiert*, indem jeder Knoten durch seinen **string-value** ersetzt wird.

string-value eines Knotens

XPath definiert für jede Knotenart einen **string-value**

- Wurzel: Konkatenation der **string-value**s der Kinder
- Element: Konkatenation der **string-value**s der Kinder
- Attribut: Wert des Attributs
- Text: Character Data
- Namespace: die URI
- Verarbeitungsanweisungen: *daten*
- Kommentare: Inhalt

Kommentare und Variablen

- Kommentare (können geschachtelt werden)

```
(: this is a comment :)
```

- Variablenreferenzen
  - `$variable` oder `$nsp:variable` (mit NS-Präfix)
  - Bindung im Kontext oder durch `for`-Ausdruck (später)
  - enthalten beliebige Werte

Arithmetische Ausdrücke

- mit den üblichen Operatoren `+`, `-`, `*`, `div`
- für Integers: `idiv`, `mod`
- Vorzeichen `-`

Behandlung von Folgen als Argumente

- falls ein Argument die leere Folge ist, so ist das Ergebnis die leere Folge
- falls alle Argumente ein-elementige Folgen von Zahlen sind, so liefert die Operation eine ein-elementige Ergebnisfolge
- anderenfalls Laufzeitfehler

Folgenausdrücke

$$\langle\, exp \,\rangle_1, \langle\, exp \,\rangle_2, \ldots, \langle\, exp \,\rangle_n$$

- berechnet die Konkatenation der Werte von $\langle\, exp \,\rangle_1$,..., $\langle\, exp \,\rangle_n$
- leere Folge, falls $n = 0$
- keine geschachtelten Folgen möglich

  `(1, (2, 3, 4), ((5)), (), (((6,7),8),9))`

  hat den gleichen Wert wie

  `1, 2, 3, 4, 5, 6, 7, 8, 9`
- weitere Alternative

  `1 to 9`

Folgenoperationen

- Folgen von Knoten können mit Mengenoperatoren bearbeitet werden
  - `union` oder `|`
  - `intersect`
  - `except`

  Diese Operationen entfernen Duplikate und liefern Ergebnis in Dokumentenordnung
- Jede Folge kann als Startpunkt für die Auswertung eines Pfadausdrucks dienen.

```
(fn:doc("veggie.xml"), fn:doc("bbq.xml"))//rcp:title
```

Filterausdrücke

- Generalisierung von Prädikaten auf beliebige Folgen
- Syntax

$$\langle exp \rangle_1 [\langle exp \rangle_2]$$

- Kontext ist Element der Folge $\langle exp \rangle_1$
- das Element ist in $\langle exp \rangle_2$ durch "." verfügbar
- Bsp

```
(30 to 60)[. mod 5 = 0 and position()>20]
```

liefert

```
50, 55, 60
```

Vergleiche Drei Arten von Vergleichen mit unterschiedlichen und teilweise unerwarteten Eigenschaften bzw. Nicht-Eigenschaften.

- Wertvergleiche
- allgemeine Vergleiche
- Knotenvergleiche

Zusätzlich gibt es den strukturellen Vergleich über Funktion `fn:deep-equal`

Wertvergleich

- Vergleich von Atomen
- Operatoren `eq`, `ne`, `lt`, `le`, `gt`, `ge`

Ablauf der Vergleichsoperation

- Beide Argumente werden atomisiert.
- Wenn ein Argument die leere Folge ist $\Rightarrow$ leere Folge.
- Wenn ein Argument Länge > 1 hat, so ist das Ergebnis `false()`.
- Argumente haben inkompatiblen Typ $\Rightarrow$ Laufzeitfehler.
- Anderenfalls wird der Vergleich durchgeführt.

Bsp (alle `true()`)

```
8 eq 4+4
//rcp:description/text() eq "Some recipes used in the XML tutorial."
(//rcp:ingredient)[1]/@name eq "beef cube steak"
```

Allgemeiner Vergleich

- liefert immer ein Ergebnis
- Operatoren =, !=, <, <=, >, >=

Ablauf der Vergleichsoperation

- Die Argumente werden atomisiert zu Folgen X und Y.
- Falls es eine Paar (x, y) von Elementen $x \in X$ und $y \in Y$ gibt, die in der gewünschten Relation liegen, so ist das Ergebnis `true()`.
- Anderenfalls ist das Ergebnis `false()`.

Bsp (alle `true()`)

```
8 = 4+4
(1,2) = (2,4)
//rcp:ingredient/@name = "salt"
```

Knotenvergleich

- Vergleicht Knoten bezüglich Identität und Dokumentenordnung
- Operatoren `is`, <<, >>

Ablauf der Vergleichsoperation

- Wenn ein Argument die leere Folge ist, dann ist das Ergebnis die leere Folge.
- Wenn beide Argumente ein-elementige Folgen von Knoten sind, so wird der entsprechende Test durchgeführt und das Ergebnis zurückgegeben.
- Anderenfalls wird ein Laufzeitfehler ausgelöst.

Bsp (alle `true()`)

```
(//rcp:recipe)[2] is //rcp:recipe[rcp:title/text() eq "Ricotta Pie"]
/rcp:collection << (//rcp:recipe)[4]
(//rcp:recipe)[4] >> (//rcp:recipe[3])
```

Vergleich der Vergleiche Die Vergleiche liefern oft verschiende Ergebnisse bei Anwendung auf die gleichen Argumente.

Bsp: Die `ingredients` 40 und 53 seien beide `salt`, aber mit unterschiedlichen Mengen.

```
((//rcp:ingredient)[40]/@name, (//rcp:ingredient)[40]/@amount) eq
((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[53]/@amount)
```

$\Rightarrow$ `false`.

```
((//rcp:ingredient)[40]/@name, (//rcp:ingredient)[40]/@amount) =
((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[53]/@amount)
```

$\Rightarrow$ `true`.

```
((//rcp:ingredient)[40]/@name, (//rcp:ingredient)[40]/@amount) is
((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[53]/@amount)
```

$\Rightarrow$ Laufzeitfehler.

Eigenschaften der Vergleiche Eine Ordnungsrelation sollte reflexiv, transitiv und antisymmetrisch sein. Außerdem sollten zwei Werte entweder gleich oder ungleich sein.

- Allgemeiner Vergleich ist weder reflexiv, noch transitiv, noch antisymmetrisch, noch ist `!=` die Negation von `=`
- Wertvergleich ist transitiv und antisymmetrisch, aber weder reflexiv noch gilt die Negation
  bei Einschränkung der Argumente auf ein-elementige Folgen gelten alle Eigenschaften
- Knotenvergleich ist nicht reflexiv, aber transitiv und antisymmetrisch

Funktionen

- Kontext definiert
  - Namespaces für Funktionen
  - Signaturen dazu
- Namespace für Standardfunktionen (106 Funktionen)

  `http://www.w3.org/2005/02/xpath-functions`

  Default Namespace mit Präfix `fn`
- Namespace `xs` für Datenkonstruktion und -umwandlung

  `http://www.w3.org/2001/XMLSchema`

Aufruf von Funktionen

- Standardsyntax: `f(x1,...,fn)`
- Auswertungsreihenfolge der Argument implementierungsabhängig
- Achtung beim Funktionsaufruf mit einem Folgenargument

  `fn:avg(1,2,3,4)`

  übergibt der Funktion vier Argumente (und schlägt fehl)

  `fn:avg((1,2,3,4))`

  übergibt ein Argument, welches eine Folge ist (berechnet den Durchschnitt der Folge)
- Siehe `http://www.w3.org/TR/xpath-functions/` für Beschreibung aller Funktionen

For-Ausdruck

`for $`$\langle name \rangle$ `in` $\langle exp \rangle_1$ `return` $\langle exp \rangle_2$

- Wertet $\langle exp \rangle_1$ zu einer Folge F aus
- Wertet $\langle exp \rangle_2$ für jedes Element von F aus
- Dabei ist `$`$\langle name \rangle$ jeweils das Element von F (innerhalb des `return` Ausdrucks $\langle exp \rangle_2$)
- Die Ergebnisse werden konkateniert

Beispiele

- Einfache Zutaten zählen

```
for $r in //rcp:recipe
  return fn:count($r//rcp:ingredient[fn:not(rcp:ingredient)])
```

  liefert

```
11, 12, 15, 8, 30
```

- Verschachtelte Schleife

```
for $i in (1 to 5)
  for $j in (1 to $i)
    return $j
```

  liefert

```
1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5
```

Bedingter Ausdruck

if ($\langle exp \rangle_1$) then $\langle exp \rangle_2$ else $\langle exp \rangle_3$

Beispiel:

```
fn:avg(
  for $r in //rcp:ingredient return
    if ( $r/@unit = "cup" )
       then xs:double($r/@amount) * 237
    else if ( $r/@unit = "teaspoon" )
       then xs:double($r/@amount) * 5
    else if ( $r/@unit = "tablespoon" )
       then xs:double($r/@amount) * 15
    else ()
)
```

Ausdrücke mit Quantoren

some $⟨*name*⟩ in ⟨*exp*⟩$_1$ satisfies ⟨*exp*⟩$_2$

every $⟨*name*⟩ in ⟨*exp*⟩$_1$ satisfies ⟨*exp*⟩$_2$

- liefern boolesche Ergebnisse
- mit offensichtlicher Bedeutung

Beispiel

```
some $r in //rcp:ingredient satisfies $r/@name eq "sugar"
```

Quantorenelimination

```
some $r in //rcp:ingredient satisfies $r/@name eq "sugar"
```

ist äquivalent zu

```
((for $r in //rcp:ingredient return
    if ($r/@name eq "sugar") then
      fn:true() else () ),
 fn:false())[1]
```

Ähnliche Elimination für every möglich.

10.2.10 Schlussbemerkung zu XPath

Typen: Bisher nur die *ungetypte* Version von XPath betrachtet. Falls ein XML Schema für Dokument vorliegt, konvertiert die Atomisierung in XML Schema-Typen und es sind Typtests zur Laufzeit möglich

XPath 1.0 vs 2.0: Viele Implementierungen stellen nur XPath 1.0 zur Verfügung

- Funktionsbibliothek von 1.0 ist wesentlich kleiner (und Aufruf ohne Namespace-Präfix)
- XPath 1.0 führt stillschweigend viele (z.T. verwirrende) Typkonversionen durch, dies geschieht in 2.0 explizit durch `xs:` Operatoren
- XPath 2.0 ist Teilmenge von XQuery

10.3 XML Pointer Language (XPointer)

Ziel: Sprache zur Adressierung von Dokumentfragmenten (Ziele von Links)

- Erweiterung von XPath
- Punkte und Bereiche adressierbar
- Stringsuche
- Einsatz als Fragmentbezeichner in URIs:
  - `dokument#id` (abkürzende Form)
  - `dokument#schema(...)` (schemabasierte Form)

XPointer-Ausdrücke benutzen Sonderzeichen, die in XML-Attributen oder in URIs nicht verfügbar sind. $\Rightarrow$ übliche Kodierung `>` bzw. %20.

10.3.1 XPointer Adressierung

$$
\begin{array}{lcl}
\langle \mathit{XPointer} \rangle & ::= & \langle \mathit{BareName} \rangle \\
& | & \langle \mathit{ChildSeq} \rangle \\
& | & \mathit{FullXPtr} \\
\langle \mathit{FullXPtr} \rangle & ::= & \langle \mathit{XPtrPart} \rangle^{+} \\
\langle \mathit{XPtrPart} \rangle & ::= & \langle \mathit{Scheme} \rangle (\langle \mathit{Expr} \rangle) \\
\langle \mathit{Scheme} \rangle & ::= & \texttt{xpointer} \mid \ldots \\
\langle \mathit{ChildSeq} \rangle & ::= & \texttt{/1}(\texttt{/}[0-9]^{*})^{*} \\
& | & \langle \mathit{Name} \rangle (\texttt{/}[0-9]^{*})^{+}
\end{array}
$$

Beispiele

`xpointer(id("introduction"))`

Element mit `ID="introduction"`;
Abkürzung (*bare name*): `introduction`

`element(/4/7` für das siebte Kind vom vierten Kind vom Wurzelelement (*child sequence*)

`xpointer(id("introduction")) xpointer(child::*[4]/child::*[7])`

Definition Ein *Ort* ist

- ein Baumknoten eines Dokuments (wie XPath)
- ein Punkt im Dokument (ein Baumknoten und ein Index)
- ein Bereich zwischen zwei Punkten im Dokument

Neue ⟨*NodeType*⟩

⟨*NodeType*⟩	::=	`point`	falls Punkt
	|	`range`	falls Bereich
	|	...	

Neue Funktionen

```
xpointer(id("chap1")/range-to(id("chap2")))
```

```
xpointer(descendant::REVST/range-to(following::REVEND[1]))
```

```
string-range(//title,"Thomas Pynchon")[17]
```

Abkürzung: `//` steht für `/descendant-or-self::node()/`

Patentprobleme ...

- Sun hält ein Patent auf eine Methode zur Adressierung von Dokumententeilen durch angehängte Strings und das automatische Scrollen eines Browsers auf eine so markierte Stelle
- Siehe `http://www.xml.com/pub/a/2001/01/17/xpointer.html`
- Mittlerweile gelöst

10.4 XML Linking Language (XLink)

Verallgemeinerung der Hyperlink-Fähigkeiten von HTML
(wohlbekannt und erprobt aus anderen Hypertext-Systemen)

- Links mit mehr als zwei Teilnehmern
- Assoziation von Daten mit einem Link
- Externe Link-Datenbanken

Hyperlinks

Was ist ein Hyperlink?

Eine Relation zwischen Teilen von Dokumenten, möglichst in einem Benutzerinterface darstellbar.

Eigenschaften von `<a href=...>` in HTML

- Ziel des Links gegeben durch URI
- Link befindet sich an seinem Ursprung
- Link identifiziert sein Ziel
- Nur von Ursprung nach Ziel zu verwenden
- Art der Verwendung festgelegt.

10.4.1 Der `xlink` Namespace

```
<rootElement
   xmlns:xlink="http://www.w3.org/1999/xlink">
   ...
 </rootElement>
```

enthält *globale Attribute* (benutzbar mit jedem Element): `type`, `href`, `role`, `title`, `show`, `actuate`, `from`, `to`

```
<local:link
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:local="http://uni-freiburg.de/"
   xlink:type="simple"
   xlink:href="faculties.xml"
   xlink:role="facultylist" xlink:title="Liste der Fakultäten"
   xlink:show="new" xlink:actuate="onRequest">
   Liste der Fakultäten
</local:link>
```


10.4.2 Einfache Links

vom Typ `xlink:type="simple"` verbinden eine lokale und eine externe Ressource

```
 <!ELEMENT studentlink ANY>
 <!ATTLIST studentlink
   xlink:type    (simple)             #FIXED "simple"
   xlink:href    CDATA                #IMPLIED
   xlink:role    NMTOKEN              #FIXED "student"
   xlink:title   CDATA                #IMPLIED
   xlink:show    (new|replace
                 |embed
                 |undefined)          #FIXED "replace"
   xlink:actuate (onLoad|onRequest
                 |undefined)          #FIXED "onRequest"
 >
...
 <studentlink
   xlink:href="/students/mmueller.xml"
   xlink:title="Wer ist Michael Müller?">Michael</studentlink>
 organisiert die Fête.
```


Globale Attribute eines `simple`-Link

- `xlink:href="`*URI*`"`
  Zeiger auf externe Ressource
- `xlink:role="`⟨*qualified-name*⟩`"` (semantisch)
  zur Gruppierung und Bennennung von Ressourcen in traversal rules
- `xlink:title="`⟨*Lesbarer Titel des Links*⟩`"`
  (semantisch)

- `xlink:show` (Verhalten)

Wert	Verhalten
`new`	neues Browserfenster
`replace`	benutze altes Browserfenster
`embed`	einbetten in aktuelles Browserfenster
`undefined`	beliebig

- `xlink:actuate` (Verhalten)

Wert	Aktivierung
`onLoad`	beim Laden des Links
`onRequest`	beim Anklicken
`undefined`	beliebig

10.4.3 Erweiterte Links

- verbinden eine beliebige Anzahl von internen oder externen Ressourcen
- *out-of-line*, falls alle Ressourcen extern, sonst *inline*
- *out-of-line* erlaubt schreibgeschützte Ressourcen
- Verbindung zunächst ohne Richtung ($\rightarrow$ traversal rules)
- Getrennte Spezifikation von
  - Teilnehmern am Link
  - Benutzung des Links
  - Einbettung des Links

- Element wird zum erweiterten Link durch Attribut `xlink:type="extended"`
- enthält beliebige Elemente mit
  - `xlink:type="locator"`
    Adresse einer externen Ressource
    muß `xlink:href` angeben
    ggf. `xlink:role`
  - `xlink:type="arc"`
    traversal rule
    Quellenrolle spezifiziert durch `xlink:from`
    Zielrolle(n) spezifiziert durch `xlink:to`
  - `xlink:type="title"`
    für menschliche Benutzer
  - `xlink:type="resource"`
    interne Ressource
    ggf. `xlink:role`

  als (direkte) Kinder

Beispiel/DTD

```
<!-- courseload = extended-type
     tooltip = title-type
     go = arc-type
     student, course, audit, advisor = locator-type
     gpa = resource-type
-->
<!ELEMENT courseload
  (tooltip*, (student|course|audit|advisor|go*), gpa)>
<!ATTLIST courseload
  xlink:type          (extended)   #FIXED "extended"
  xlink:role          NMTOKEN      #FIXED "courseload"
  xlink:title         CDATA        #IMPLIED
>
```

Beispiel/Dokument

```
<courseload
  xlink:title="Course Load for Pat Jones">

  <go xlink:from="gpa" xlink:to="course" />
  <go xlink:from="student" xlink:to="course" />
  <go xlink:from="student" xlink:to="audit" />
  <go xlink:from="student" xlink:to="advisor" />
  <student xlink:href="..." />
  <course xlink:href="..." />
  <course xlink:href="..." />
  <audit xlink:href="..." />
  <advisor xlink:href="..." />
  <gpa>3.5</gpa>
</courseload>
```

Beispiel/traversal rule

```
<!-- extendedlink = extended-type
      loc = locator-type
 -->
 <extendedlink>
   <loc xlink:href="..." xlink:role="parent" xlink:title="p1" />
   <loc xlink:href="..." xlink:role="parent" xlink:title="p2" />
   <loc xlink:href="..." xlink:role="child"  xlink:title="c1" />
   <loc xlink:href="..." xlink:role="child"  xlink:title="c2" />
   <loc xlink:href="..." xlink:role="child"  xlink:title="c3" />

   <!-- go = arc-type -->
   <go xlink:from="parent" xlink:to="child" />
 </extendedlink>
```

Beispiel/typischer Link

```
<!-- go = arc-type -->
<!ELEMENT go ANY>
<!ATTLIST go
  xlink:type    (arc)                 #FIXED "arc"
  xlink:from    NMTOKEN               #IMPLIED
  xlink:to      NMTOKEN               #IMPLIED
  xlink:show    (new|replace|embed
                |undefined)           #IMPLIED
  xlink:actuate (onLoad|onRequest
                |undefined)           #IMPLIED
  xlink:role    NMTOKEN               #IMPLIED
  xlink:title   CDATA                 #IMPLIED
>

<go
  xlink:from="student"
  xlink:to="advisor"
  xlink:show="new"
  xlink:actuate="onRequest" />
```